Theoretical Computer Science 133 (1994) 361–385
Elsevier



# Towards efficient induction mechanisms in database systems*

Jiawei Han

*School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, Canada*

*Abstract*

Han, J., Towards efficient induction mechanisms in database systems, Theoretical Computer Science 133 (1994) 327–351.

With the wide availability of huge amounts of data in database systems, the extraction of knowledge in databases by efficient and powerful induction or knowledge discovery mechanisms has become an important issue in the construction of new generation database and knowledge-base systems. In this article, an attribute-oriented induction method for knowledge discovery in databases is investigated, which provides an efficient, set-oriented induction mechanism for extraction of different kinds of knowledge rules, such as characteristic rules, discriminant rules, data evolution regularities and high level dependency rules in large relational databases. Our study shows that the method is robust in the existence of noise and database updates, is extensible to knowledge discovery in advanced and/or special purpose databases, such as object-oriented databases, active databases, spatial databases, etc., and has wide applications.

## 1. Introduction

The startling achievements of the researches and developments of database systems in the past two decades have set a solid foundation for the development of the next generation information management systems, which, according to many people's views, are data-intensive knowledge-base systems [23, 25].

Such data-intensive knowledge-base systems should store huge amounts of data and knowledge. A simple fact of life is that huge amounts of data are already being

*Correspondence to:* J. Han, School of Computing Science, Simon Fraser University, Burnaby, B.C., V5A 1S6, Canada. Email: han@cs.sfu.ca.

* The research was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Centre for Systems Science of Simon Fraser University. This paper is a substantially extended and revised version of one section of the article. Efficient deduction and induction: key to the success of data-intensive knowledge-base systems, appearing in *Formal Methods in Databases and Software Engineering*, Workshops in Computing (Springer, London, 1993).


*SSDI* 0304-3975(94)00056-O

and will continue to be collected in a large number of databases, by various kinds of data gathering systems, from satellite telemetry systems, to manufacture line instrumentation, to scanned texture and image platforms [3]. On the other hand, current technologies rely more or less on users or domain experts to manually input *knowledge*. Such kind of knowledge could be biased, error-prone, and costly (i.e., with limited productivity). With the rapid growth in size and number of available databases in commercial, industrial, administrative and other applications [7], it is necessary and interesting to examine how to extract knowledge automatically from the huge amount of data. By knowledge discovery in databases, large databases will serve as a rich, reliable source for knowledge generation and verification, and the discovered knowledge can be applied to information management, query processing, decision making, process control and many other applications. Thus it is no wonder that *knowledge discovery in databases* (or *data mining*) has been ranked recently as one of the most important research topics in 1990s by both database and machine learning researchers [23, 21].

In our previous studies [4, 11, 12, 10], an attribute-oriented induction method has been developed for knowledge discovery in relational databases. The method integrates a machine learning paradigm, especially *learning-from-examples* techniques, with database operations and extracts generalized data from actual data in databases. In this paper, we investigate the theoretical foundation of the attribute-oriented induction method and examine the extension of the mechanism to advanced and/or special purpose databases, such as object-oriented databases, active databases, and spatial databases.

The paper is organized as follows. In Section 2, the philosophical considerations of knowledge discovery in databases are discussed. In Section 3, the principles of attribute-oriented induction mechanisms are introduced. In Section 4, the algorithms for attribute-oriented induction are introduced and their computational complexities are analyzed. In Section 5, we discuss the extension of the method to different kinds of databases. Our discussion is summarized in Section 6.

## 2. Philosophical considerations for knowledge discovery in databases

There are different philosophical considerations on knowledge discovery in databases (KDD) [7, 28], which may lead to different methodologies in the development of KDD techniques. To focus our study on some effective and promising mechanisms, the following assumptions, based on our philosophical reasoning, have been made in our discussion of knowledge discovery in databases.

**Assumption 2.1.** A database stores a large amount of information-rich, relatively reliable and stable data.

It is commonly recognized that a database stores a large amount of data with rich information. However, data in a database may be incomplete, redundant, incorrect, or

highly dynamic in certain applications [28], which makes knowledge discovery a challenging task. The assumption on data stability and reliability facilitates the development of knowledge discovery mechanisms firstly in relatively simple situations and then evolution of the techniques step-by-step towards more complicated ones.

**Assumption 2.2.** A database usually stores only *positive* training examples for a learning process but not *negative* ones.

Many machine learning algorithms use positive training examples to generalize hypotheses and negative examples to specialize them in induction processes. In database-oriented induction, the inference of negative examples by the *closed world assumption* [22] simply based on the nonexistence of certain data is of limited use because a nonexistent instance does not imply that it cannot occur in the database but simply indicates that it has not yet occurred (but may occur in the future).

Since only positive training examples are stored in a database, an induction process relies mainly on the generalization process. Thus, generalization should be performed conservatively to avoid potential over-generalization.

**Assumption 2.3.** A knowledge discovery process is initiated by a user's learning request.

Idealistically, one may expect that a knowledge discovery system will perform interesting discovery autonomously without human interaction. However, since learning can be performed in many different ways on any subset of data in the database, huge amounts of knowledge may be generated from even a medium sized database by unguided, autonomous discovery, whereas much of the discovered knowledge could be out of user's interests. In contrast, a command-driven discovery may lead to the guided discovery with a focus on the interested set of data and therefore represents relatively constrained search for the desired knowledge. Thus, command-driven discovery is adopted in this study.

**Assumption 2.4.** Generalized rules are expressed in terms of high level concepts.

Without concept generalization, discovered knowledge is expressed in terms of primitive data (data stored in the databases), often in the form of functional or multivalued dependency rules or primitive level integrity constraints. On the other hand, with concept generalization, discovered knowledge can be expressed in terms of concise, expressive and higher level abstraction, in the form of generalized rules or generalized constraints, and be associated with statistical information. Obviously, it is often desirable for large databases to have rules expressed at the concept levels higher than the primitive ones.

**Assumption 2.5.** Background knowledge is generally available for knowledge discovery process.

Discovery may be performed with the assistance of relatively strong background knowledge (such as conceptual hierarchy information, etc.) or with little support of background knowledge. The discovery of conceptual hierarchy information itself can be treated as a part of a knowledge discovery process. However, the availability of relatively strong background knowledge not only improves the efficiency of a discovery process but also expresses user's preference for guided generalization, which may lead to an efficient and desirable generalization process. Thus we have this assumption.

Following these assumptions, our mechanism for knowledge discovery in database can be outlined as follows. First, a knowledge discovery process is initiated by a learning request, which is usually in relevance to only a subset of data in a database. A data retrieval process is initiated to collect the set of relevant data. Second, generalization is performed on the set of retrieved data using the background knowledge and a set of generalization operators. Third, the generalized data is simplified and transformed into a set of generalized rules, which may facilitate querying database knowledge, intelligent query answering, decision making, process control, and many other applications.

A major challenge for learning in databases is computational efficiency. To overcome this difficulty, an attribute-oriented induction method [4, 11] has been developed in our study which strives for efficiency in two aspects: (i) knowledge-directed learning, and (ii) attribute-oriented induction. The former is achieved by providing knowledge about the learning task, data relevance, expected rule forms and concept hierarchies. The latter is achieved by an attribute-oriented concept tree ascension technique, which performs generalization attribute-by-attribute until the relevant data is generalized to a certain level and the size of the generalized relation is reasonably small. Then, more flexible induction methods can be applied to the relatively small set of (generalized) data according to different criteria, such as quantitative measurement, expert knowledge, user-performance, and others. The method substantially reduces the search space and improves the efficiency of a database learning process.

In the following sections, we first analyze the method for knowledge discovery in relational database systems [11] and then extend the method to knowledge discovery in advanced database systems.

## 3. Principles of attribute-oriented induction in relational databases

### *3.1. Primitives for the specification of a learning task*

Three primitives are provided for the specification of a learning task: *task-relevant data*, *background knowledge*, and *expected representation of learning results.*

*3.1.1. Data relevant to the discovery process*

A database usually stores a large amount of data, of which only a portion may be relevant to a particular learning task. Data relevance may extend over several relations in a relational database. A relational query can be used to collect a set of task-relevant data from one or a set of relations. Such a data collection process results in a new data relation, called *initial* (*working*) *relation*, for the learning task.

An initial relation, $p(A_1, A_2, \ldots, A_n)$, consists of a set of $n$ attributes $A_1, A_2, \ldots, A_n$. Suppose each attribute $A_i$ is defined on the domain of $D_i$. The initial relation $p(A_1, A_2, \ldots, A_n)$ must be defined as a subset of $D_1 \times \cdots \times D_n$.

Each tuple in the initial relation $p$ can be viewed as a positive example in an induction process. Undoubtedly, *learning-from-examples* [18] should be an important strategy for knowledge discovery in databases. As explained in Assumption 2.2, there are usually no *negative* (training) examples stored in the database. Thus, *generalization* on an initial relation, and subsequently on intermediate generalized relations, should be performed cautiously to avoid over-generalization.

*3.1.2. Background knowledge*

A concept hierarchy defines a sequence of mappings from a set of lower-level concepts to their higher-level correspondences. A concept hierarchy can be defined on one or a set of attribute domains. Suppose a hierarchy $H$ is defined on a set of domains $D_i, \ldots, D_k$ in which different levels of concepts are organized into a taxonomy. The concept taxonomy is usually partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by a reserved word "ANY") whereas the most specific concepts correspond to the specific values of attributes in the database. Formally, we have

$$H: \quad D_i \times \cdots \times D_k \Rightarrow H_l \Rightarrow H_{l-1} \Rightarrow \cdots \Rightarrow H_0, \tag{3.1}$$

where $H_l$ represents the set of concepts at level $l$ (one level higher than those at the primitive level), $H_{l-1}$ represents the concepts at one level higher than those at $H_l$, etc., and $H_0$, the highest level hierarchy, may contain solely the most general concept, "ANY".

A concept hierarchy defines the mapping rules from a set of lower-level concepts to their corresponding higher level ones. Such mappings are often data- or application-specific. A concept hierarchy can be provided by knowledge engineers or domain experts. This is reasonable even for large databases since a concept hierarchy registers only the *distinct* discrete attribute values or ranges of numerical values for an attribute which are, in general, not very large and can be input by domain experts. Many concept hierarchies, such as *birthplace* (*city*, *province*, *country*), are actually stored in the database implicitly, which can be made explicit by specifying certain attribute mapping rules. Also, many concept hierarchies can be discovered automatically and/or refined dynamically based on the data distribution statistics among different attribute values [6, 11], which is especially effective for numerical attributes.

Concept hierarchies represent necessary background knowledge which directs the generalization process. Using a concept hierarchy, the rules learned can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

Different concept hierarchies can be constructed on the same attribute based on different view-points or preferences. For example, the birthplace could be organized according to administrative regions, geographic locations, size of cities, etc. Usually, a commonly referenced concept hierarchy is associated with an attribute as the default one. Other hierarchies can be chosen explicitly by preferred users in a learning process. Also, it is sometimes preferable to perform induction in parallel along more than one concept hierarchy and determine an appropriate representation based on later generalization results.

*3.1.3. Representation of learning results*

An induction process may discover different kinds of rules, such as characteristic rules, discriminant rules, data evolution regularities, etc. A *characteristic rule* is an assertion which characterizes a concept satisfied by all or most of the examples in the class undergoing learning (called the *target class*). For example, the symptoms of a specific disease can be summarized by a characteristic rule. A *discriminant rule* is an assertion which discriminates a concept of the class being learned (the *target class*) from other classes (called *contrasting classes*). For example, to distinguish one disease from others, a discriminant rule should summarize the symptoms that discriminate this disease from others. A *data evolution regularity* rule is an assertion which describes the general properties of a set of data in the database which change with time.

From a logical point of view, each tuple in a relation is a logic formula in conjunctive normal form, and a data relation is characterized by a large set of disjunctions of such conjunctive forms. Thus, both the data for learning and the rules discovered can be represented in either relational form or first-order predicate calculus.

A relation which represents *intermediate* (or *final*) learning results is called an *intermediate* (or a *final*) *generalized relation*. In a generalized relation, some or all of its attribute values are generalized data, that is, nonleaf nodes in the concept hierarchies. An attribute in a (generalized) relation is at a *desirable* level if it contains only a small number of distinct values in the relation. A user or an expert may like to specify a small integer as a *desirable attribute threshold* for an attribute. Such a threshold can also be set as a default by the system. In this case, an attribute is at the *desirable* level if it contains no more distinct values than its attribute threshold. Moreover, the attribute is at the *minimum desirable level* if it would contain more distinct values than the threshold when it were specialized to a level lower than the current one. A special generalized relation $R'$ of an initial relation $R$ is the *prime relation* of $R$ if every attribute in $R'$ is at the minimum desirable level.

Some learning-from-examples algorithms require the final learned rule to be in conjunctive normal form [18]. This requirement may not be reasonable for large

databases since the generalized data often contain different cases. However, a rule containing a large number of disjuncts indicates that it is in a complex form and further generalization should be performed. Therefore, the final generalized relation should be represented by either one tuple (a conjunctive rule) or a small number of tuples corresponding to a disjunctive rule with a small number of disjuncts. A system may allow a user to specify the preferred *generalization threshold* (or *generalized relation threshold*), a maximum number of disjuncts of the resulting formula. For example, if the threshold value is set to five, the final generalized rule will consist of *at most* five disjuncts.

Exceptional data often occur in a large relation. It is important to consider exceptional cases when learning in databases. Statistical information helps learning algorithms handle exceptions and/or noisy data [19]. A special attribute, *count*, can be added to each generalized relation to register the number of tuples in the original relation which are generalized to the current tuple in the generalized relation. The attribute *count* carries database statistics and supports the pruning of scattered data and the generalization of the concepts which take a majority of count. The final generalized rule will be the rule which represents the characterstics of a *majority* number of facts in the database (called an *approximate rule*) or indicates *statistical* measurement of each conjunct or disjunct in the rule (called a *statistical rule*).

### *3.2. Attribute-oriented induction techniques*

#### *3.2.1. Basic generalization techniques*

A set of basic techniques for attribute-oriented induction in relational databases are summarized as follows [11, 13].

**Technique 3.1** (*Data focusing*). Generalization should be performed only on the set of data which are relevant to the learning request.

This technique is obvious since only the task-relevant set of data need to be studied.

**Technique 3.2** (*Generalization on the smallest decomposable components*). Generalization should usually be performed on the smallest decomposable components (or attributes) of a data relation.

**Rationale.** This technique is based on the *least commitment principle* (commitment to the minimally generalized concepts) [9] to avoid over-generalization. If the generalization were not performed on the smallest decomposable components/attributes but on larger clusters of attributes, some interesting relationships related to smaller components might have been missed, i.e., not discoverable after such generalization. However, the technique does not exclude the possibility of generalization on composite attributes if there exist such generalization operators, and if such

a generalization considers also the possibilities of generalization on the smallest decomposable components. □

**Technique 3.3** (*Attribute removal*). If there are a large set of distinct values in an attribute of the working relation, but (1) there is no generalization operator on the attribute, or (2) its higher-level concepts are expressed in another attribute, the attribute should be removed from the working relation in generalization.

**Rationale.** An attribute-value pair represents a conjunct in a generalized rule. The removal of a conjunct eliminates a constraint and thus generalizes the rule. If there is a large set of distinct values in an attribute but there is no generalization operator for it (case 1), or its higher-level concepts are expressed in another attribute (case 2), the removal of the attribute generalizes the relation, which corresponds to the generalization rule, *dropping conditions*, in *learning-from-examples* [18]. □

**Technique 3.4** (*Attribute generalization*). If there are a large set of distinct values in an attribute in the working relation, but there exist a set of generalization operators on the attribute, a generalization operator should be selected and be applied to the attribute at every step of generalization.

**Rationale.** The generalization of an attribute value using a selected generalization operator makes the object cover more cases than the original one and thus generalizes the concept. This corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [18]. □

Usually, generalization corresponds to the ascension of the concept hierarchy if the generalization is to express a concept by its corresponding more generalized one in a given concept hierarchy. As a result, different tuples may be generalized to equivalent ones where two (generalized) tuples are *equivalent* if they have the same corresponding attribute values without considering their special internal attribute *count*, which registers the number of tuples in the initial relation that are generalized to the current tuple. Notice that a *generalized tuple* is a carrier of the general properties of a set of initial tuples. The *count* accumulated in the generalized relation incorporates quantitative information in the learning process.

**Technique 3.5** (*Count propagation*). The value of the count of a tuple should be carried to its generalized tuple, and the count should be accumulated when merging equivalent tuples in generalization.

This technique is based on the definition of *count* and the merge of equivalent tuples.

**Technique 3.6** (*Attribute generalization control*). Generalization on an attribute $a_i$ is performed until the concepts in $a_i$ has been generalized to a desired level, or the number of distinct values in $a_i$ in the resulting relation is no greater than a prespecified or default attribute threshold.

This technique is based on the desirability of representation of each attribute at its desired level or by a small number of distinct values.

Formally, the generalization of an attribute $p$ of a tuple $t$ can be written in an abstract way as $Gen(t.p)$, where *Gen* is an abstract relational generalization operator which can be transformed into a concrete operation based on the role of the attribute and a specific learning requirement. Moreover, the generalized attribute of a tuple can be further generalized by applying *Gen* again, which could be the same or different generalization operators compared with the one applied in the last generalization. If the generalization is performed by applying the same sequence of generalization operators on a component $p$ of $t$, we should have

$$Gen^{n-1}(Gen(t.p)) \equiv Gen(Gen^{n-1}(t.p)). \tag{3.2}$$

For example, a person $P_1$'s address can be generalized from a concrete address, such as the number of a street into a street block, then a street, a district, a city, a province, a country, etc. when necessary by applying the generalization operator *Gen* several times, such as $Gen(Gen(\cdots Gen(P_1.address)\cdots))$.

A generalization operator *Gen* can be applied to an attribute $a_i$ on every tuple in a working relation $\mathscr{W}_k$, resulting in a new generalized relation $\mathscr{R}_k$. Thus , a *database generalization operator, DBGen,* is introduced which applies the tuple generalization operator *Gen* to an attribute $a_i$ of *every* tuple in the working relation. That is,

$$\begin{aligned}\mathscr{R}_k = DBGen(\mathscr{W}_k, a_i) = \{t' &: t \in \mathscr{W}_k \wedge t'.a_i \\ &= Gen(t.a_i) \wedge \forall (j \neq i) t'.a_j = t.a_j\}\end{aligned} \tag{3.3}$$

For example, for a working relation $\mathscr{W}_0 = Person$ in a universal database, $DBGen(\mathscr{W}_0, address)$ derives a resulting relation $\mathscr{R}_0$ with one-step generalization on the attribute "*address*" (e.g., from street number to street block) and all the other attributes unaltered.

The application of a database generalization operator *DBGen* on a *working relation* results in a more general *resulting relation,* which in turn becomes the working relation in the next round of database generalization. Such a generalization process proceeds until the concept in every attribute in the resulting relation has reached to a desired concept level, or the number of distinct values in every attribute is no greater than its attribute threshold. The generalized resulting relation so obtained is called a *prime (generalized) relation.*

Notice that it is often unnecessary to derive the prime relation by performing *DBGen* step-by-step on a working relation as described above, which may take many

passes in generalization. This is because by analysis of the set of distinct values of each attribute in the initial relation, generalization can be performed on the initial relation in one step to derive the set of generalized values in the prime relation as if a set of *DBGen*'s were performed several times on each single attribute. By doing so, the generalization from an initial relation to the prime relation will only need to be performed in a single pass.

*3.2.2. Generalized rule extraction*

Since the above induction process enforces only attribute generalization control, the prime generalized relation so extracted may still contain a relatively large number of generalized tuples. Two alternatives can be developed for the extraction of generalized rules from a prime generalized relation: (1) further generalize the prime relation to derive a *final generalized relation* which contains no more tuples than a prespecified relation threshold, and then extract the final generalized rule; and (2) directly extract *generalized feature table* and present feature-based multiple rules.

Alternative 1 is based on Technique 3.7.

**Technique 3.7** (*Relation generalization control*). Generalization on a prime generalized relation is performed until the number of distinct generalized tuples in the resulting relation is no greater than a prespecified relation threshold.

At this stage, there are usually alternative choices for selecting a candidate attribute for further generalization. The interestingness of the final generalized rule relies on the selection of the attributes to be generalized and the selection of generalization operators. Such selections can be based on data semantics, user preference, generalization efficiency, etc. Many techniques developed in previous studies on machine learning [19], statistics [16], fuzzy set and rough set theories [27], etc. can be applied to the selection of attributes and operators. Criteria, such as the preference of a larger reduction ratio on the number of tuples or the number of distinct attribute values, the simplicity of the final learned rules, etc., can also be used for selection. Interesting rules can often be discovered by following different paths leading to several generalized relations for examination, comparison and selection. Following different paths corresponds to the way in which different people may learn differently from the same set of examples. The generalized relations can be examined by users or experts *interactively* to filter out trivial rules and preserve interesting ones [28]. After this generalization, final generalized rules(s) can be extracted from a final generalized relation, where a tuple in the generalized relation is transformed to conjunctive normal form, and multiple tuples are transformed to disjunctive normal form.

Alternative 2 takes the set of generalized tuples and maps them into a generalized feature table. Based on the generalized feature table, multiple generalized feature-based rules can be presented. An example of such a generalized feature table is presented in Section 3.4 with a detailed derivation algorithm in [13].

## 4. Attribute-oriented induction: algorithms and experiments

### *4.1. Basic attribute-oriented induction algorithm*

The above techniques can be summarized into the following generalization algorithm which extracts generalized characteristic rules in a relational database based on a user's learning request.

**Algorithm 4.1** (*Basic attribute-oriented induction in relational database*). Discovery of a set of generalized characteristic rules in a relational database based on a user's learning request.

*Input.* (i) A relational database *DB*, (ii) a database query specification, *DBQuery*, related to the learning task, (iii) $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes $a_i$, and (iv) $T$, a *relation threshold*, and $T_i$, a set of *attribute thresholds* for attributes $a_i$.

*Output.* A characteristic rule based on the learning request.

*Method.*

1. *InitRel.* Derivation of the *initial working relation*, $\mathscr{W}_0$. This is done by collecting the set of task-relevant data by execution of the relational query *DBQuery* on the database, *DB*, according to the learning request.

2. *PreGen.* Preparation of the generalization process. This is performed by (1) scanning the initial working relation $\mathscr{W}_0$ once and collecting the distinct values for each attribute $a_i$ and the number of occurrences of each distinct value in $\mathscr{W}_0$, (2) computing the minimum desired level $L_i$ for each attribute $a_i$ based on its given or default attribute threshold $T_i$, as explained further in the following paragraph, and (3) determining the mapping-pairs $(v, v')$ for each attribute $a_i$ in $\mathscr{W}_0$, where $v$ is a distinct value of $a_i$ in $\mathscr{W}_0$, and $v'$ is its corresponding generalized value at level $L_i$.

Notice that the minimum desirable level $L_i$ of $a_i$ is determined based on a sequence of *Gen* operators and/or the available concept hierarchy so that all of the distinct values for attribute $a_i$ in $\mathscr{W}_0$ can be generalized to a small number $\tau$ of distinct generalized concepts, where $\tau$ is the largest possible number of distinct generalized values of $a_i$ in $\mathscr{W}_0$ at a level of concept hierarchy which is no greater than the attribute threshold of $a_i$. Also, notice that a concept hierarchy, if given, can be adjusted or refined dynamically, or, if not given, can be generated dynamically when possible (e.g., for numerical values), based on data distribution statistics.

3. *PrimeGen.* Derivation of the *prime generalized relation*, $\mathscr{R}_p$. This is done by (1) replacing each value $v$ in $a_i$ of $\mathscr{W}_0$ with its corresponding superordinate concept $v'$ determined at the *PreGen* stage; and (2) merging equivalent tuples in the working relation with *count* accumulated. The resulting relation is $\mathscr{R}_p$.

4. *RuleGen.* Presentation of generalized rules.

- Determine which of the two alternatives: (1) *final generalized relation*, or (2) *generalized feature table*, should be chosen in the presentation of generalized rules. This can be predetermined by experts or determined by interaction with users.

- *Case 1*: Alternative 1 is chosen. If the total number of tuples in $\mathscr{R}_p$ is within the *relation threshold* $T$, $\mathscr{R}_p$ is the final generalized relation $\mathscr{R}_f$. Otherwise, further generalization is performed on $\mathscr{R}_p$ by selection of certain attributes for further generalization. This process continues until the number of distinct generalized tuples is no greater than $T$. The final generalized relation $\mathscr{R}_f$ can be mapped to a *final generalized rule* for output.
- *Case 2*: Alternative 2 is chosen. $\mathscr{R}_p$ is mapped to a *generalized feature table*, which can be further mapped to a set of generalized rules, if desired, for output.

Step 1 of the algorithm is essentially a relational query whose processing efficiency depends on the query processing algorithm. The cost of Steps 2 and 3 (i.e., from the initial working relation $\mathscr{W}_0$ to the prime relation $\mathscr{R}_p$) is dominant in the processing of the remaining steps since Step 4 works on a much smaller relation $\mathscr{R}_p$ in comparison with the initial relation $\mathscr{W}_0$. The following theorem presents the processing efficiency of Steps 2 and 3.

**Theorem 4.1.** *The worst-case time complexity of Steps* 2 *and* 3 *in Algorithm* 4.1 *is* $O(n \times p)$ *where* $n$ *is the number of tuples of the initial relation* $\mathscr{W}_0$, *and* $p$ *is the number of tuples of the prime relation* $\mathscr{R}_p$.

**Proof.** The total processing cost is the accumulation of the cost of the following three parts.

*Part 1* collects the statistics of the initial relation, which scans the initial relation $\mathscr{W}_0$, collects the distinct attribute values in $\mathscr{W}_0$, and registers the number of occurrences of each distinct value in $\mathscr{W}_0$. This step takes $O(n)$ time since the process scans the initial relation exactly once.

*Part 2* computes the minimum desired level and determines the mapping pairs $(v, v')$ for each attribute. Suppose there are $a$ relevant attributes, the average number of distinct values for each attribute is $m$, the average number of levels to be climed up (or the number of *Gen* operators to be applied) is $l$, and the average cost of tree-climbing or application of *Gen* operator is $c$. Then the cost of Part 2 should be $a \times m \times l \times c$, which is in the order of $O(n)$ since $m \leqslant n$, and $a$, $l$, and $c$ are small constants.

*Part 3* derives the prime relation by merging equivalent tuples, which is performed as follows. For each tuple $t$ in $\mathscr{W}_0$, substitute its attribute values based on value mapping-pairs derived in Part 2. This results in a generalized tuple $t'$. If $t'$ is not already in the prime relation $\mathscr{R}_p$, insert it into $\mathscr{R}_p$ and set its *count* to 1. Otherwise, increment the *count* of $t'$ in $\mathscr{R}_p$. Since there are total $n$ tuples in $\mathscr{W}_0$ and $p$ tuples in $\mathscr{R}_p$, the cost of such substitution and searching will be $O(n \times p)$.

Summing-up the cost of the three portions, it is clear that the worst case time complexity should be $O(n \times p)$. □

Notice that since the size of the prime relation is usually small, in the order of $\log(n)$, the total processing cost of generalization takes about $O(n \log(n))$, which is efficient for large databases.

### *4.2. Algorithm implementation and experiments*

Based upon the attribute-oriented induction technique, a prototyped experimental database learning system, DBLEARN, has been constructed [13]. The system, DBLEARN, takes learning requests as inputs, applies the knowledge discovery algorithm(s) on the data stored in a database, with the assistance of the concept hierarchy information stored in a concept hierarchy base. The learning requests are specified in the syntax similar to SQL, a standard relational database language. The outputs of the system are generalized relations or knowledge rules extracted from the database. The system is implemented in C with the assistance of UNIX software packages LEX and YACC (for compiling the DBLEARN language interface) and operates in conjunction with the SyBase DBMS software. A database learning language for DBLEARN is specified in an extended BNF grammar.

Experimentation using DBLEARN has been conducted on several large real databases, including *the NSERC Grants Information system*, which contains the information about the research grants awarded by NSERC (*the Natural Sciences and Engineering Research Council of Canada*) in the year of 1990–1991. The database consists of 6 large data relations. The central relation table, *award*, contains 10 087 tuples with 11 attributes.

The background knowledge in DBLEARN is represented by a set of concept hierarchies. In each hierarchy, the most general concept is the null description (described by a reserved word "ANY"), and the most specific concepts correspond to the specific values of attributes in the database. Fig. 1 shows the concept hierarchy for provinces in Canada, where $A \subset B$ indicates that $B$ is a generalization of $A$. Notice that since the three provinces *B.C.*, Ontario, and Quebec take most of research grants, the three provinces will be automatically promoted to the top level in the learning process by dynamically refinement of the concept hierarchy by the system.

Other concept hierarchies, such as $\{1 \ldots 19\,999\} \subset$ 1–20 K, $\{20\,000 \ldots 39\,999\} \subset$ 20–40 K, ..., $\{26\,000 \ldots 26\,499\} \subset$ AI (where 26 000, ..., 26 499 represent NSERC discipline codes), are also stored in the concept hierarchy table. Notice that concept hierarchies for numerical attributes, such as amount, can be generated automatically based on data distribution statistics, which is also implemented in the DBLEARN system as an alternative. Many learning requests have been posed to this database during our experimentation. Interesting knowledge rules/relationships about NSERC

---

{Alberta, Saskatchewan, Manitoba} ⊂ Prairies
{British Columbia, Prairies} ⊂ Western Canada
{Ontario, Quebec} ⊂ Central Canada
{New Brunswick, Nova Scotia, Newfoundland, Prince_Edward_Island} ⊂ Maritime
{Western Canada, Central Canada, Maritime, Territories} ⊂ ANY(province)

---

Fig. 1. A concept hierarchy for attribute *province*.

research grant awards in relevance to geographic locations, research fields, etc. have been discovered by our experimentation. One such experimental example is illustrated as follows.

**Example 4.1.** Let the query be to discover a characteristic rule for NSERC support of operating grants for AI (Artificial Intelligence) researchers in relevance to the geographical locations, the number of grants and the amount distribution of the grants in 1990 to 1991. The learning task is presented in DBLEARN as follows.

**learn characteristic rule for** "AI_OP_Grants"
**from** *award*
**where** *disc_code* = "AI" and *grant_code* = "Operating_Grants"
**in relevance to** *amount*, *province*, *percentage*(*count*), *percentage*(*amount*)

Notice that *percentage* (*attribute*) is a built-in function which returns the *percentage* of the summation of the numerical *attribute* values in the generalized tuple divided by the summation of the same *attribute* values in the whole generalized relation.

When the query is posed to the system, relevant data are collected by data retrieval from the Grant Information Database. Then attribute-oriented induction is performed on the collected data. The learning result of the query is presented in Table 1. The row "*amount* = 20–40 K, *geo_area* = B.C., *percentage*(*num_of_grants*) = 12.7%, *and percentage*(*amount*) = 16.3%" indicates that for the Operating Grants in AI in the amount between $20 000 and $39 999, B.C. researchers take 12.7% of the total number of grants and 16.3% of the total amount of grants. The last row contains the summary information of the entire generalized relation. Some negligible proportion (less than

Table 1
A generalized relation for AI Operating Grants

| *amount* | *geo_area* | *percentage*(*num_of_grants*) | *percentage*(*amount*) |
|---|---|---|---|
| 1–20 K | B.C. | 5.6% | 4.1% |
| 1–20 K | Prairies | 15.5% | 10.3% |
| 1–20 K | Quebec | 14.1% | 9% |
| 1–20 K | Ontario | 25.3% | 17.8% |
| 1–20 K | Maritime | 2.8% | 1% |
| 20–40 K | B.C. | 12.7% | 16.3% |
| 20–40 K | Prairies | 5.6% | 6.2% |
| 20–40 K | Ontario | 9.8% | 13% |
| 20–40 K | Maritime | 1.4% | 1.7% |
| 40–60 K | B.C. | 1.4% | 4% |
| 40–60 K | Ontario | 4.2% | 11.3% |
| >60 K | Quebec | 1.4% | 4.2% |
| $1 464 250 | Canada | 99.8% | 98.9% |

1%) of the AI operating grants scattered across Canada is ignored in the table. Thus, the total number of grants in the table takes 99.8% of the total available AI operating grants.

The relationships between *geo_area*, *number_of_grants*, *amount_of_grants*, *amount_category*, etc. can also be presented in extracted feature tables as shown in Table 2, when necessary, using the extracted prime relation. The system interacts with users for explicit instructions on the necessity of such a presentation.

The performance of the DBLEARN system is satisfactory. The response time of the above query (including the SyBase data retrieval time) is less than 20 seconds on an SUN/SPARC II workstation.

### *4.3. Discussion*

#### *4.3.1. A comparison with other machine learning methods*

Algorithm 4.1 introduces a simple and efficient way to extract characteristic rules in relational databases. The major difference of this induction algorithm from the previously developed *learning-from-examples* algorithms [5, 19] is attribute-oriented vs. tuple-oriented induction. It is essential to compare these two approaches.

Both tuple-oriented and attribute-oriented induction take attribute removal and concept tree ascension as their major generalization techniques. However, the former technique performs generalization tuple by tuple, whereas the latter, attribute by attribute.

The tuple-oriented approach examines the training examples one at a time to induce generalized concepts [9, 20]. In order to discover the most specific concept that is satisfied by all of the training examples, the algorithm must search every node in the search space which represents the possible concepts derived from the generalization on this training example. Since different attributes of a tuple may be generalized to different levels, the number of nodes to be searched for a training example may involve a huge number of possible combinations.

Table 2
A generalized feature table for AI operating grants

| *geo_area* | *amount* | | | | sum |
|---|---|---|---|---|---|
| | 1–20 K | 20–40 K | 40–60 K | >60 K | |
| Ontario | 18 | 7 | 3 | 0 | 28 |
| B.C. | 4 | 9 | 1 | 0 | 14 |
| Prairies | 11 | 4 | 0 | 0 | 15 |
| Maritime | 2 | 1 | 0 | 0 | 3 |
| Quebec | 10 | 0 | 0 | 1 | 11 |
| Total | 45 | 21 | 4 | 1 | 71 |

On the other hand, an attribute-oriented algorithm performs generalization on each attribute uniformly for all the tuples in the data relation at the *early* generalization stages. It essentially isolates its consideration to individual attributes, or factored version spaces [24] (but not their combinations) at such stages. Factoring the version space may substantially improve the computational efficiency. Suppose there are $k$ concept hierarchies used in the generalization and there are $p$ nodes in each concept hierarchy. The total size of $k$ factorized version spaces is $p \times k$. However, the size of the unfactorized version space for the same concept tree is $p^k$ [24].

Notice that an algorithm which explores different possible combinations for different attributes in a large number of tuples during the early generalization stages cannot be fruitful since such combinations will be merged in further generalizations. Different possible combinations should be explored only when the relation has been generalized to a relatively small prime relation.

Another obvious advantage of our approach over many other learning algorithms is our integration of the learning process with database operations. In contrast to most existing learning algorithms which do not take full advantages of these database facilities [5, 19], our approach primarily adopts relational operations, such as selection, join, projection (extracting relevant data and removing attributes), tuple substitution (ascending concept trees), and intersection (discovering common tuples among classes). Since relational operations are set-oriented and have been implemented efficiently in many existing systems, our approach is not only efficient but easily exported to many relational systems.

*4.3.2. Learning discriminant rules*

Since a discriminant rule distinguishes the concepts of the target class from those of contrasting classes, the generalized condition in the target class that overlaps the condition in contrasting classes should be detected and be removed from the description of a discriminant rule. Thus, a discriminant rule can be extracted by generalizing the data in both the target class and the contrasting classes synchronously and excluding properties that overlap in both in the final generalized rule. This is illustrated in the following technique.

**Technique 4.1** (*Handling overlapping tuples*). If there are overlapping tuples in both the target and contrasting classes, these tuples should be marked and be excluded from the final discriminant rule.

**Rationale.** Since the overlapping tuples represent the same disjuncts in both the target class and the contrasting class(es), the concept described by the overlapping tuples cannot be used to distinguish the target class from the contrasting class(es). By detecting and marking overlapping tuples, we have the choice of including only those assertions which have a discriminant property in the rule, which ensures the correctness of the learned discriminant rule. □

Based on Technique 4.1 and the principles of attribute-oriented induction, the discovery of discriminant rules can be performed as follows. Firstly, the prime relation corresponding to the initial relation in the target class, called *prime target relation*, can be extracted by the attribute-oriented induction method, which executes essentially the same algorithm as Algorithm 4.1. Secondly, the concepts of the initial relation(s) in the contrasting class(es) are generalized to the same level as those in the prime target relation, which results in a *prime contrasting relation*. Thirdly, compare the tuples in the prime target relation against those in the prime contrasting relation, and mark those tuples which overlap the two prime relations. If the number of unmarked tuples in the prime target relation is within the specified threshold, this set of unmarked tuples represents the learned discriminant rule. Otherwise, further generalization of the two prime relations should be performed to generalize the concepts to a satisfactory level.

When further generalizing the two prime relations, overlapping marks should be inherited in their generalized tuples because two tuples, if overlapped in the two prime relations, will still overlap by further generalization. Moreover, since generalization may produce new overlapping tuples, an overlapping check should be performed at each step of generalization. The generalization process may repeat until the number of unmarked tuples in the target class is below the specified threshold value.

Notice that a discriminant rule provides a sufficient condition but may not be a necessary one for an object (or a tuple) to be in the target class. Although those tuples which meet the condition are in the target class, those not necessarily satisfy the condition may also be in the target class since the rule may not cover *all* of the positive examples of the target class in the database.

Similar to learning characteristic rules, the discriminative feature of a target class can be described quantitatively by a *quantitative discriminant rule*. Learning quantitative discriminant rules by attribute-oriented induction has been presented in [12], which will not be discussed further.

### *4.3.3. Learning data evolution regularities*

Data evolution regularity reflects the trend of changes in a database over time. Although both database schema and database contents may change over time, we assume that the database schema remains stable in data evolution in order to focus our attention to database contents.

Let a *database instance*, $DB_t$, be a database state (all of the data in the database) at time $t$. The discovery of data evolution regularity requires that there exists at least two different database instances, $DB_{t_1}$ and $DB_{t_2}$, where $t_1 \neq t_2$. Notice that the results on this view can be easily transformed to other representations of temporal data [26], such as different temporal values being represented within the same tuple but associated with different timestamps.

Similar to knowledge discovery in static data, data evolution regularities can be classified into characteristic rules and discriminant rules, where the former summarize characteristics of the changed (or changing) data; while the latter distinguish general

characteristics of the relevant data in the database, such as $DB_{t_1}$, from those in another database instance, such as $DB_{t_2}$.

The discovery of general data evolution regularity needs to first extract evolving data from different database instances (or data with different timestamps) which can be performed by database queries with certain temporal conditions. Then the extracted data can be grouped as one initial working relation in the case of learning characteristic rules, or be partitioned into target classes and contrasting classes in the case of learning discriminant rules, or be partitioned into a sequence of groups corresponding to certain time sequences in the case of discovering the trend of data evolution. The generalization and rule extraction process for discovering these rules are similar to that for discovering the corresponding rules in stable data.

#### *4.3.4. Learning approximate rules and incremental learning*

As shown in our previous discussion, attribute-oriented induction can learn disjunctive rules and handle exceptional cases elegantly by incorporating statistical information, e.g., by *count* accumulation, in the learning process. The association of the *count* information with each disjunct (i.e., each generalized tuple) leads naturally to learning approximate rules [19, 21], because the conditions with negligible weight (proportion) can be dropped in further generalization or rule derivation since they will have minimal influence of the final conclusion.

This feature also facilitates incremental learning in large databases. For example, when a new tuple is inserted into a data relation, instead of restarting the learning process from scratch, it is preferable to amend and fortify what was learned from the previous data. Our algorithms can be easily extended to facilitate such *incremental learning* [19]. Let the generalized relation be stored in the database. When a new tuple is inserted into a database, the concepts of the new tuple are first generalized to the level of the concepts in the generalized relation. Then the generalized tuple can be naturally merged into the generalized relation.

Furthermore, with the association of *count* information, data sampling and parallelism can be explored in knowledge discovery. Attribute-oriented induction can be performed by *sampling* a subset of data from a huge set of relevant data or by first performing induction *in parallel* on several partitions of the relevant data set and then merging the generalized results.

#### *4.3.5. Dynamic refinement of concept hierarchies*

Our study assumes that there exists relatively strong background knowledge in the form of concept hierarchies for knowledge discovery. A given hierarchy may often need to be modified or refined dynamically based on user's learning requests and/or the statistics of the relevant set of data. For example, if the learning request is to analyze the birth place of the students of Simon Fraser University, the top level concepts could be: {B.C, other_provinces_in_Canada, foreign}; however, if it is to analyze the birth place of the *faculty* of the University, the appropriate top level

concepts could be {North_America, Europe, Asia, other_countries}. Both concept hierarchies can be obtained by dynamic adjustment of a given concept hierarchy based on the analysis of the statistical distribution of the relevant data sets.

Generalization on numerical attributes can be performed similarly but in a more automatic way by the examination of data distribution characteristics [6]. In many cases, it may not require any predefined concept hierarchies. For example, the ages of the graduate students in a university can be clustered into several groups, such as {below 23, 23–26, 25–30, over 30}, according to a relatively uniform data distribution criteria or using some statistical clustering analysis tools. Appropriate names can be assigned to the generalized numerical ranges, such as {very young, young, ...} by users or experts to convey more semantic meaning.

Moreover, concept "hierarchies" can be in many different shapes. They can be organized as lattices, unbalanced trees, general directed acyclic graphs, etc. For example, if the concepts are organized as a lattice, some single concepts may be generalized to more than one concept. These generalized concepts can be put into intermediate generalized relations upon which further generalizations can be performed as discussed. As a consequence, the size of intermediate generalized relations may increase at some stage in the generalization process because of the effect of a lattice. However, since the generalization is controlled by a specified threshold value, the intermediate generalized relation will eventually shrink in subsequent generalizations.

## 5. Towards knowledge discovery in advanced database systems

In this section, we briefly examine the extension of attribute-oriented induction to knowledge discovery in advanced and/or special purpose databases, such as object-oriented databases, active databases, and spatial and multimedia databases.

### *5.1. Attribute-oriented induction in object-oriented databases*

An OODB [15, 1] organizes a large set of complex data objects into classes which are in turn organized into class/subclass hierarchies with rich data semantics. Each object in a class is associated with (1) an object-identifier, (2) a set of attributes which may contain sophisticated data structures, set- or list-valued data, class composition hierarchies, multimedia data, etc., and (3) a set of methods which specify the computational routines or rules associated with the object class.

Attribute-oriented induction in object-oriented databases can be performed by first generalizing a set of complex data components into relatively simple generalized concepts and then compiling these high-level concepts into intermediate generalized relations to which the relational knowledge discovery technique such as attribute-oriented induction applies.

To facilitate generalization of complex data components, it is important to implement efficiently a relatively small set of generalization operators on complex data objects. We examine the generalization of different components of complex objects, including object identifiers, unstructured and structure values, class composition hierarchies, inherited and derived data, methods specified by rules or procedures, etc.

First, we examine the *generalization of object identifiers*. Since objects belong to certain classes which in turn are organized into certain class/subclass hierarchies, an object identifier can be first generalized to its corresponding lowest subclass name which can in turn be generalized to a higher level class/subclass name by climbing up the class/subclass hierarchy.

Secondly, we examine the *generalization of structured data*. The generalization of single valued, numerical and nonnumerical data are studied in the previous sections. The generalization of complex structure-valued data, which are popular in OODBs, such as set-valued and list-valued data and data with nested structures, can be explored in several ways in order to extract interesting patterns from such data sets.

Take set-valued attributes as an example. A set-valued attribute may be of homogeneous or heterogeneous types, which can be typically generalized in two ways: (1) generalization of each value in a set into its corresponding higher level concepts, or (2) derivation of the general behavior of a set, such as the number of elements in the set, the types or value ranges in the set, the weighted average for numerical data, etc. For example the *hobby* of a person is a set-valued attribute which contains a set of values, such as {tennis, hockey, chess, violin, nintendo}, which can be generalized into a set of high level concepts, such as {*sports*, *music*, *computer_games*}, or into 5 (the number of hobbies in the set), or both, etc. Moreover, a *count* can be associated with a generalized value to indicate how many elements are generalized to the corresponding generalized value, such as {sports(3), music(1), computer_games(1)}, where sports(3) indicates three kinds of sports, etc.

Thirdly, we examine the *generalization on inherited properties*. In our OODB, some attribute or method of an object class may not be explicitly specified in the class itself but are inherited from its higher level classes. Some OODB systems may allow *multiple inheritance* (i.e., the properties to be inherited from more than one superclass) when the class/subclass "hierarchy" is organized in the shape of a lattice. The inherited properties of an object can be derived by query processing in the OODB. From the knowledge discovery point of view, it is unnecessary to distinguish which data are stored within the class and which are inherited from its superclass. As long as the set of relevant data are collected by query processing, the knowledge discovery process will treat the inherited data in the same way as the data stored in the object class and perform generalization accordingly.

Fourthly, we examine the *generalization of methods*. Method is an important component of OODBs, which is usually defined by a computational procedure/function or by a set of deduction rules. Many behavioral data of objects can be derived by application of methods. However, it is difficult to perform generalization on the method itself unless the generalization of the method is clearly understood by

application programmers and is coded as a new method which directly performs the required generalization. In general, the generalization on the data derived by method application should be performed in two steps: (1) deriving the task-relevant set of data by application of the method and, possibly, also data retrieval; and (2) performing generalization by treating the derived data as the existing ones.

Finally, we examine the *generalization on class composition hierarchies.* An attribute of an object may be composed by another object, and some of whose attributes may be composed in turn by other objects, thus forming a class composition hierarchy. Generalization on a class composition hierarchy can be viewed as generalization on a set of (possibly infinite, if the nesting is recursive) nested structured data.

In principle, the reference to a composite object may traverse via a long sequence of references along the corresponding class composition hierarchy. However, in most cases, the longer sequence of references is traversed, the weaker semantic linkage between the original objects and the referenced composite objects is obtained. For example, one attribute "*vehicles_owned*" of an object class "*student*" could refer to another object class "*car*" which may contain an attribute "*auto_dealer*", which may refer to its "*manager*" with an attribute "*children*". Obviously, it is unlikely to find interesting regularities between a group of students and their car dealers' managers' children.

Therefore, generalization on a class of objects should be mainly performed on its own descriptive attribute values, methods, etc. with limited references to (and therefore generalization on) its composed object hierarchy. That is, in order to discover relatively interesting knowledge, generalization should be performed only on the composite objects closely related to the currently focused class(es) but not on those which have only remote and rather weak semantic linkages.

### *5.2. Attribute-oriented induction in spatial and multimedia databases*

Based on the similar methodology of knowledge discovery in object-oriented databases, attribute-oriented induction can be performed on spatial and multimedia databases.

Besides concept tree ascension and structured data summarization, aggregation and approximation should be considered as an important means of generalization, which is especially useful for generalization of attributes with large sets of values, spatial or multimedia data, etc.

Take spatial data as an example. It is desirable to generalize detailed geographic points into clustered regions, such as business, residential, industry, or agricultural areas, according to the land usage. Such generalization often requires the merge of a set of geographic areas by spatial operations, such as spatial union, or spatial clstering algorithms. Approximation is an important technique in such generalization: In spatial merge, it is necessary not only to merge the regions of similar types within the same general class but also to ignore some scattered regions with different types if they are unimportant to the study. For example, different pieces of land for different

purposes of agricultural usage, such as vegetables, grain, fruits, etc. can be merged into one large piece of land by spatial merge. However, such an agricultural land may contain highways, houses, small stores, etc. If the majority land is used for agriculture, the scattered spots for other purposes can be ignored, and the whole region can be claimed as an agricultural area by approximation. The spatial operators, such as *spatial_union*, *spatial_overlapping*, *spatial_intersection*, etc., which merge scattered small regions into large, clustered regions can be considered as generalization operators in spatial aggregation and generalization.

A multimedia database may contain hypertext, graphics, images, maps, voice, music, and other forms of audio/video information. Such multimedia data are typically stored as sequences of bytes with variable lengths, and segments of data are linked together for easy reference. Generalization on multimedia data can be performed by recognition and extraction of the essential features and/or general patterns of such data.

There are many ways to extract the essential features or general patterns from segments of multimedia data. For an image, the size and color of the contained objects or the major regions in the image can be extracted by aggregation and/or approximation. For a segment of music, its melody can be summarized based on the approximate patterns that repeatedly occur in the segment and its style can be summarized based on its tone, tempo, major musical instruments played, etc. For an article, its abstract or general organization such as the table of contents, the subject and index terms frequently occurring in the article, etc. may serve as generalization results. In general, it is a challenging task to generalize multimedia data to extract the interesting knowledge implicitly stored in the data. Further research should be devoted to this issue.

With such generalization of spatial data and multimedia data, attribute-oriented induction mechanism can then be applied to the extraction of different kinds of rules at the higher concept levels in a similar way to that in relational databases.

### 5.3. *Towards knowledge discovery in active databases*

Finally, attribute-oriented induction can be applied to knowledge discovery in active databases [14], which is outlined as follows.

Active databases [2, 17, 8] react to the changes of environments or production processes automatically by referencing database data and triggering appropriate actions. In a dynamic environment, most data are presented at low, primitive levels and are generated rapidly, continuously, dynamically, in huge volumes. Moreover, process control often requires *prompt*, *real-time*, and *intelligent* reactions in response to situation changes in the environment.

In such situation, knowledge discovery techniques can be integrated with data sampling techniques and active database techniques by sampling the data, extracting general regularities of a dynamic system, and triggering appropriate actions when some unusual situation is detected by the knowledge discovery routines.

With the help of knowledge discovery techniques, clear and concise relationships or regularities among the collected data can be discovered and expressed at a high concept level. This facilitates the understanding of large and low-level processing data and the interaction with high level active database rules, which may lead to prompt, real-time, and intelligent reactions to the changes of the environment.

Since attribute-oriented induction is an important technique in knowledge discovery in different kinds of data, and an active database contains essentially the same kind of data, but in dynamic, active environments, it is obvious that attribute-oriented induction should be an important candidate technique for knowledge discovery in active databases.

## 6. Conclusions

In this paper, an efficient induction mechanism, the attribute-oriented induction, is investigated for knowledge discovery in large databases. The method provides an efficient, set-oriented induction mechanism for extraction of different kinds of knowledge rules, including characteristic rules, discriminant rules, data evolution regularities and high level dependency rules in large databases.

Our study shows that the method is sound, efficient, robust in the existence of noise and database updates, and is extensible to knowledge discovery in advanced and/or special purpose databases, including object-oriented databases, active databases, spatial databases, etc., and thus leads to wide applications in knowledge acquisition in large databases, construction of knowledge-bases from large databases, querying database knowledge, cooperative query answering, semantic query optimization, process control, etc.

Knowledge discovery is an important and promising direction in the development of data and knowledge-base systems. Our preliminary study of the induction mechanism leads to an efficient implementation of the DBLEARN system in relational databases. We are working on the integration of the attribute-oriented induction method with advanced database technologies and will report our progress in the future.

## Acknowledgment

The author would like to express his thanks to Yongjian Fu, Yue Huang and Yandong Cai for their implementations of the DBLEARN system and their experiments of the attribute-oriented induction method for knowledge discovery in large relational databases. Also, the author would like to express his thanks to them and to Nick Cercone, Hiroyuki Kawano, Wei Lu, Raymond Ng, Shojiro Nishio and Beng Chin Ooi for their discussions on the research issues related to knowledge discovery in databases.

## References


[1] F. Bancilhon, C. Delobel and P. Kanellakis, *Building an Object-Oriented Database System: The story of O2* (Morgan Kaufmann, Los Altos, CA, 1992).

[2] C. Beeri and T. Milo, A model for active object-oriented database, in: *Proc. 17th Internat. Conf. on Very Large Data Bases* (Barcelona, Spain, September 1991) 337–349.

[3] R.J. Brachman, Viewing data from a knowledge representation lens, in: *Proc. Internat. Conf. on Building and Sharing of Very Large-Scale Knowledge Bases'93* (Tokyo, Japan, December 1993) 117–120.

[4] Y. Cai, N. Cercone and J. Han, Attribute-oriented induction in relational databases, in: G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases* (AAAI/MIT Press, 1991) 213–228.

[5] T.G. Dietterich and R.S. Michalski, A comparative review of selected methods for learning from examples, in: R.S. Michalski et al., ed., *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, (Morgan Kaufmann, Los Altos, CA, 1983) 41–82.

[6] D. Fisher, Improving inference through conceptual clustering, in: *Proc. 1987 AAAI Conf.* (Seattle, Washington, July 1987) 461–465.

[7] W.J. Frawley, G. Piatetsky-Shapiro and C.J. Matheus, Knowledge discovery in databases: An overview, in: G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases* (AAAI/MIT Press, 1991) 1–27.

[8] N.H. Gehani, H.V. Jagadish and O. Shmueli, Composite event specification in active databases: Model and implementation, in: *Proc. 18th Internat. Conf. on Very Large Data Bases* (Vancouver, Canada, August 1992) 327–338.

[9] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence* (Morgan Kaufmann, Los Altos, CA, 1987).

[10] J. Han, Efficient deduction and induction: Key to the success of data-intensive knowledge-base systems, in: V.S. Alagar, L.V.S. Lakshmanan and F. Sadri, eds., *Formal Methods in Databases and Software Engineering* (Springer, Berlin, 1993) 139–157.

[11] J. Han, Y. Cai and N. Cercone, Knowledge discovery in databases: An attribute-oriented approach, in: *Proc. 18th Internat. Conf. on Very Large Data Bases* (Vancouver, Canada, August 1992) 547–559.

[12] J. Han, Y. Cai and N. Cercone, Data-driven discovery of quantitative rules in relational databases, *IEEE Trans. Knowledge and Data Engineering* **5** (1993) 29–40.

[13] J. Han, Y. Cai, N. Cercone and Y. Huang, DBLEARN: A knowledge discovery system for databases, in: *Proc. 1st Internat. Conf. on Information and Knowledge Management* (Baltimore, MD, November 1992) 473–481.

[14] J. Han, S. Nishio and H. Kawano, Knowledge discovery in object-oriented and active databases, in: *Proc. Internat. Conf. on Building and Sharing of Very Large-Scale Knowledge Bases'93* (Tokyo, Japan, December 1993) 205–214.

[15] W. Kim, *Introduction to Object-Oriented Databases* (MIT Press, Cambridge, MA, 1990).

[16] D.J. Lubinsky, Discovery from database: A review of AI and statistical techniques, in: *Proc. IJCAI-89 Workshop on Knowledge Discovery in Databases* (Detroit, MI, August 1989) 204–218.

[17] D.R. McCarthy and U. Dayal, The architecture of an active database management system, in: *Proc. 1989 ACM SIGMOD International Conference on Management of Data* (Portland, OR, June 1989) 215–24.

[18] R.S. Michalski, A theory and methodology of inductive learning, in: Michalski et al. ed., *Machine Learning: An Artificial Intelligence Approach, Vol. 1* (Morgan Kaufmann, Los Altos, CA, 1983) 83–134.

[19] R.S. Michalski, J.G. Carbonell and T.M. Mitchell, *Machine Learning, An Artificial Intelligence Approach, Vol. 2* (Morgan Kaufmann, Los Altos, CA, 1986).

[20] T.M. Mitchell, Generalization as search, *Artificial Intelligence* **18** (1982) 203–226.

[21] G. Piatetsky-Shapiro and W.J. Frawley, *Knowledge Discovery in Databases* (AAAI/MIT Press, 1991).

[22] R. Reiter, Towards a logical reconstruction of relational database theory, in: M. Brodie, J. Mylopoulos and J. Schmidt, eds., *On Conceptual Modeling* (Springer, Berlin, 1984) 191–233.

[23] A. Silberschatz, M. Stonebraker and J.D. Ullman, Database systems: Achievements and opportunities, *Comm. ACM* **34** (1991) 94–109.

[24] D. Subramanian and J. Feigenbaum, Factorization in experiment generation, in: *Proc. 1986 AAAI Conf.* (Philadelphia, PA, August 1986) 518–522.

[25] J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Vols. 1 and 2* (Computer Science Press, Rockville, MD, 1989).

[26] G.T.J. Wuu and U. Dayal, A unified model for temporal object-oriented databases, in: *Proc. of 8th Internat. Conf. on Data Engineering* (Phoenix, AZ, February 1992) 584–593.

[27] W. Ziarko, The discovery, analysis, and representation of data dependencies in databases, in: G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases* (AAAI/MIT Press, 1991) 239–258.

[28] J. Zytkow and J. Baker, Interactive mining of regularities in databases, in: G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases* (AAAI/MIT Press, 1991) 31–54.